Table of Contents

Tensorflow Tutorials

> From: pkmital/tensorflow_tutorials

UPDATE (July 12, 2016)

New **free MOOC course** covering all of this material in much more depth, as well as much more including combined variational autoencoders + generative adversarial networks, visualizing gradients, deep dream, style net, and recurrent networks: **https://www.kadenze.com/courses/creative-applications-of-deep-learning-with-tensorflow-i/info**

TensorFlow Tutorials

You can find.mdthon source code under the .mdthon `directory, and associated notebooks under` notebooks`.

	Source code	Description
1	**basics**	Setup with tensorflow and graph computation.
2	**linear_regression**	Performing regression with a single factor and bias.
3	**polynomial_regression**	Performing regression using polynomial factors.
4	**logistic_regression**	Performing logistic regression using a single layer neural network.
5	**basic_convnet**	Building a deep convolutional neural network.
6	**modern_convnet**	Building a deep convolutional neural network with batch normalization and leaky rectifiers.
7	**autoencoder**	Building a deep autoencoder with tied weights.
8	**denoising_autoencoder**	Building a deep denoising autoencoder which corrupts the input.
9	**convolutional_autoencoder**	Building a deep convolutional autoencoder.
10	**residual_network**	Building a deep residual network.
11	**variational_autoencoder**	Building an autoencoder with a variational encoding.

Installation Guides

- TensorFlow Installation
- OS specific setup
- Installation on EC2 GPU Instances

For Ubuntu users using.mdthon3.4+ w/ CUDA 7.5 and cuDNN 7.0, you can find compiled wheels under the `wheels` directory. Use `pip3 install tensorflow-0.8.0rc0.md3-none-any.whl` to install, e.g. and be sure to add: `export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda/lib64"` to your `.bashrc`. Note, this still requires you to install CUDA 7.5 and cuDNN 7.0 under `/usr/local/cuda`.

Resources

- Official Tensorflow Tutorials
- Tensorflow API
- Tensorflow Google Groups
- More Tutorials

Author

Parag K. Mital, Jan. 2016.

http://pkmital.com

License

```
"""Summary of tensorflow basics.

Parag K. Mital, Jan 2016."""
```

```
# %% Import tensorflow and pyplot
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
# %% tf.Graph represents a collection of tf.Operations
# You can create operations by writing out equations.
# By default, there is a graph: tf.get_default_graph()
# and any new operations are added to this graph.
# The result of a tf.Operation is a tf.Tensor, which holds
# the values.
```

```
# %% First a tf.Tensor
n_values = 32
x = tf.linspace(-3.0, 3.0, n_values)
```

```
# %% Construct a tf.Session to execute the graph.
sess = tf.Session()
result = sess.run(x)
```

```
# %% Alternatively pass a session to the eval fn:
x.eval(session=sess)
# x.eval() does not work, as it requires a session!
```

```
# %% We can setup an interactive session if we don't
# want to keep passing the session around:
sess.close()
sess = tf.InteractiveSession()
```

```
# %% Now this will work!
x.eval()
```

```
# %% Now a tf.Operation
# We'll use our values from [-3, 3] to create a Gaussian Distrib
ution
sigma = 1.0
mean = 0.0
z = (tf.exp(tf.neg(tf.pow(x - mean, 2.0) /
                   (2.0 * tf.pow(sigma, 2.0)))) *
     (1.0 / (sigma * tf.sqrt(2.0 * 3.1415))))
```

```
# %% By default, new operations are added to the default Graph
assert z.graph is tf.get_default_graph()
```

```
# %% Execute the graph and plot the result
plt.plot(x.eval(), z.eval())
plt.show()
```

```
# %% We can find out the shape of a tensor like so:
print(z.get_shape())
```

```
# %% Or in a more friendly format
print(z.get_shape().as_list())
```

```
# %% Sometimes we may not know the shape of a tensor
# until it is computed in the graph.  In that case
# we should use the tf.shape fn, which will return a
# Tensor which can be eval'ed, rather than a discrete
# value of tf.Dimension
print(tf.shape(z).eval())
```

```
# %% We can combine tensors like so:
print(tf.pack([tf.shape(z), tf.shape(z), [3], [4]]).eval())
```

```
# %% Let's multiply the two to get a 2d gaussian
z_2d = tf.matmul(tf.reshape(z, [n_values, 1]), tf.reshape(z, [1,
 n_values]))
```

```
# %% Execute the graph and store the value that `out` represents
 in `result`.
plt.imshow(z_2d.eval())
plt.show()
```

```
# %% For fun let's create a gabor patch:
x = tf.reshape(tf.sin(tf.linspace(-3.0, 3.0, n_values)), [n_valu
es, 1])
y = tf.reshape(tf.ones_like(x), [1, n_values])
z = tf.mul(tf.matmul(x, y), z_2d)
plt.imshow(z.eval())
plt.show()
```

```
# %% We can also list all the operations of a graph:
ops = tf.get_default_graph().get_operations()
print([op.name for op in ops])
```

```
# %% Lets try creating a generic function for computing the same
 thing:
def gabor(n_values=32, sigma=1.0, mean=0.0):
    x = tf.linspace(-3.0, 3.0, n_values)
    z = (tf.exp(tf.neg(tf.pow(x - mean, 2.0) /
                       (2.0 * tf.pow(sigma, 2.0)))) *
         (1.0 / (sigma * tf.sqrt(2.0 * 3.1415))))
    gauss_kernel = tf.matmul(
        tf.reshape(z, [n_values, 1]), tf.reshape(z, [1, n_values
]))
    x = tf.reshape(tf.sin(tf.linspace(-3.0, 3.0, n_values)), [n_
values, 1])
    y = tf.reshape(tf.ones_like(x), [1, n_values])
    gabor_kernel = tf.mul(tf.matmul(x, y), gauss_kernel)
    return gabor_kernel
```

```
# %% Confirm this does something:
plt.imshow(gabor().eval())
plt.show()
```

```
# %% And another function which can convolve
def convolve(img, W):
    # The W matrix is only 2D
    # But conv2d will need a tensor which is 4d:
    # height x width x n_input x n_output
    if len(W.get_shape()) == 2:
        dims = W.get_shape().as_list() + [1, 1]
        W = tf.reshape(W, dims)

    if len(img.get_shape()) == 2:
        # num x height x width x channels
        dims = [1] + img.get_shape().as_list() + [1]
        img = tf.reshape(img, dims)
    elif len(img.get_shape()) == 3:
        dims = [1] + img.get_shape().as_list()
        img = tf.reshape(img, dims)
        # if the image is 3 channels, then our convolution
        # kernel needs to be repeated for each input channel
        W = tf.concat(2, [W, W, W])

    # Stride is how many values to skip for the dimensions of
    # num, height, width, channels
    convolved = tf.nn.conv2d(img, W,
                             strides=[1, 1, 1, 1], padding='SAME'
)
    return convolved
```

```
# %% Load up an image:
from skimage import data
img = data.astronaut()
plt.imshow(img)
plt.show()
print(img.shape)
```

```
# %% Now create a placeholder for our graph which can store any
input:
x = tf.placeholder(tf.float32, shape=img.shape)
```

```
# %% And a graph which can convolve our image with a gabor
out = convolve(x, gabor())
```

```
# %% Now send the image into the graph and compute the result
result = tf.squeeze(out).eval(feed_dict={x: img})
plt.imshow(result)
plt.show()
```

```
"""Simple tutorial for using TensorFlow to compute a linear regr
ession.

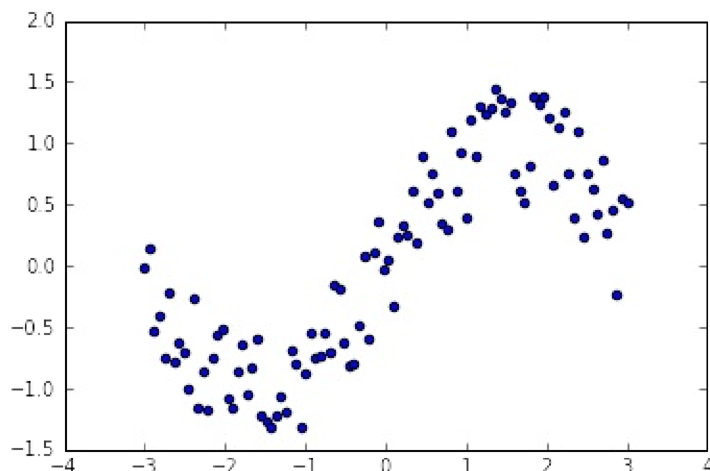
Parag K. Mital, Jan. 2016"""
```

```
'Simple tutorial for using TensorFlow to compute a linear regres
sion.\n\nParag K. Mital, Jan. 2016'
```

```
# %% imports
%matplotlib inline
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
# %% Let's create some toy data
plt.ion()
n_observations = 100
fig, ax = plt.subplots(1, 1)
xs = np.linspace(-3, 3, n_observations)
ys = np.sin(xs) + np.random.uniform(-0.5, 0.5, n_observations)
ax.scatter(xs, ys)
fig.show()
plt.draw()
```

```
/home/heythisischo/anaconda2/lib/python2.7/site-packages/matplot
lib/figure.py:397: UserWarning: matplotlib is currently using a
non-GUI backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



```
# %% tf.placeholders for the input and output of the network. Pl
aceholders are
# variables which we need to fill in when we are ready to comput
e the graph.
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
```

```
# %% We will try to optimize min_(W,b) ||(X*w + b) - y||^2
# The `Variable()` constructor requires an initial value for the
 variable,
# which can be a `Tensor` of any type and shape. The initial val
ue defines the
# type and shape of the variable. After construction, the type a
nd shape of
# the variable are fixed. The value can be changed using one of
the assign
# methods.
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
Y_pred = tf.add(tf.mul(X, W), b)
```

```
# %% Loss function will measure the distance between our observa
tions
# and predictions and average over them.
cost = tf.reduce_sum(tf.pow(Y_pred - Y, 2)) / (n_observations - 1
)
```

```
# %% if we wanted to add regularization, we could add other term
s to the cost,
# e.g. ridge regression has a parameter controlling the amount o
f shrinkage
# over the norm of activations. the larger the shrinkage, the mo
re robust
# to collinearity.
# cost = tf.add(cost, tf.mul(1e-6, tf.global_norm([W])))
```

```
# %% Use gradient descent to optimize W,b
# Performs a single step in the negative gradient
learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate).min
imize(cost)
```

```
# %% We create a session to use the graph
n_epochs = 1000
with tf.Session() as sess:
    # Here we tell tensorflow that we want to initialize all
    # the variables in the graph so we can use them
    sess.run(tf.initialize_all_variables())

    # Fit all training data
    prev_training_cost = 0.0
    for epoch_i in range(n_epochs):
        for (x, y) in zip(xs, ys):
            sess.run(optimizer, feed_dict={X: x, Y: y})

        training_cost = sess.run(
            cost, feed_dict={X: xs, Y: ys})
        print(training_cost)

        if epoch_i % 20 == 0:
            ax.plot(xs, Y_pred.eval(
                feed_dict={X: xs}, session=sess),
                    'k', alpha=epoch_i / n_epochs)
            fig.show()
            plt.draw()

        # Allow the training to quit if we've reached a minimum
        if np.abs(prev_training_cost - training_cost) < 0.000001
:
            break
        prev_training_cost = training_cost
fig.show()
```

```
1.43989
1.30686
1.18916
1.08503
0.992879
0.911325
0.839141
0.77524
0.718663
0.668564
0.624192
0.584885
0.550058
0.519194
0.491835
0.467578
0.446063
0.426977
0.410039
```

```
0.395001
0.381647
0.369783
0.359238
0.349861
0.341519
0.334094
0.327481
0.321587
0.316332
0.311643
0.307455
0.303713
0.300366
0.29737
0.294686
0.292278
0.290117
0.288175
0.286427
0.284853
0.283433
0.282151
0.280991
0.279942
0.27899
0.278125
0.277339
0.276623
0.27597
0.275373
0.274826
0.274325
0.273865
0.273442
0.273052
0.272693
0.27236
0.272052
0.271767
0.271501
0.271254
0.271024
0.27081
0.270609
0.270421
0.270245
0.27008
0.269925
0.269779
0.269642
0.269512
0.26939
```

```
0.269274
0.269165
0.269062
0.268964
0.268871
0.268783
0.268699
0.26862
0.268544
0.268472
0.268404
0.268339
0.268277
0.268218
0.268161
0.268107
0.268056
0.268007
0.26796
0.267916
0.267873
0.267832
0.267794
0.267756
0.267721
0.267687
0.267654
0.267623
0.267594
0.267565
0.267538
0.267512
0.267487
0.267464
0.267441
0.267419
0.267398
0.267378
0.267359
0.267341
0.267324
0.267307
0.267291
0.267276
0.267261
0.267247
0.267234
0.267221
0.267209
0.267197
0.267186
0.267176
0.267165
```

```
0.267156
0.267146
0.267137
0.267129
0.267121
0.267113
0.267105
0.267098
0.267092
0.267085
0.267079
0.267073
0.267067
0.267062
0.267057
0.267052
0.267047
0.267043
0.267039
0.267034
0.267031
0.267027
0.267023
0.26702
0.267017
0.267014
0.267011
0.267008
0.267006
0.267003
0.267001
0.266998
0.266996
0.266994
0.266992
0.26699
0.266989
0.266987
0.266985
0.266984
0.266982
0.266981
0.26698
0.266979
0.266978
0.266976
0.266975
0.266974



<matplotlib.figure.Figure at 0x7f8c0037e890>
```

```
"""Simple tutorial for using TensorFlow to compute polynomial re
gression.

Parag K. Mital, Jan. 2016"""
```

```
# %% Imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
# %% Let's create some toy data
plt.ion()
n_observations = 100
fig, ax = plt.subplots(1, 1)
xs = np.linspace(-3, 3, n_observations)
ys = np.sin(xs) + np.random.uniform(-0.5, 0.5, n_observations)
ax.scatter(xs, ys)
fig.show()
plt.draw()
```

```
# %% tf.placeholders for the input and output of the network. Pl
aceholders are
# variables which we need to fill in when we are ready to comput
e the graph.
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
```

```
# %% Instead of a single factor and a bias, we'll create a polyn
omial function
# of different polynomial degrees.  We will then learn the influ
ence that each
# degree of the input (X^0, X^1, X^2, ...) has on the final outp
ut (Y).
Y_pred = tf.Variable(tf.random_normal([1]), name='bias')
for pow_i in range(1, 5):
    W = tf.Variable(tf.random_normal([1]), name='weight_%d' % po
w_i)
    Y_pred = tf.add(tf.mul(tf.pow(X, pow_i), W), Y_pred)
```

```
# %% Loss function will measure the distance between our observa
tions
# and predictions and average over them.
cost = tf.reduce_sum(tf.pow(Y_pred - Y, 2)) / (n_observations -
1)
```

```
# %% if we wanted to add regularization, we could add other term
s to the cost,
# e.g. ridge regression has a parameter controlling the amount o
f shrinkage
# over the norm of activations. the larger the shrinkage, the mo
re robust
# to collinearity.
# cost = tf.add(cost, tf.mul(1e-6, tf.global_norm([W])))
```

```
# %% Use gradient descent to optimize W,b
# Performs a single step in the negative gradient
learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate).min
imize(cost)
```

```
# %% We create a session to use the graph
n_epochs = 1000
with tf.Session() as sess:
    # Here we tell tensorflow that we want to initialize all
    # the variables in the graph so we can use them
    sess.run(tf.initialize_all_variables())

    # Fit all training data
    prev_training_cost = 0.0
    for epoch_i in range(n_epochs):
        for (x, y) in zip(xs, ys):
            sess.run(optimizer, feed_dict={X: x, Y: y})

        training_cost = sess.run(
            cost, feed_dict={X: xs, Y: ys})
        print(training_cost)

        if epoch_i % 100 == 0:
            ax.plot(xs, Y_pred.eval(
                feed_dict={X: xs}, session=sess),
                    'k', alpha=epoch_i / n_epochs)
            fig.show()
            plt.draw()

        # Allow the training to quit if we've reached a minimum
        if np.abs(prev_training_cost - training_cost) < 0.000001
:
            break
        prev_training_cost = training_cost
ax.set_ylim([-3, 3])
fig.show()
plt.waitforbuttonpress()
```

```
"""Simple tutorial using code from the TensorFlow example for Re
gression.

Parag K. Mital, Jan. 2016"""
# pip3 install --upgrade
# https://storage.googleapis.com/tensorflow/mac/tensorflow-0.6.0
-py3-none-any.whl
```

```
'Simple tutorial using code from the TensorFlow example for Regr
ession.\n\nParag K. Mital, Jan. 2016'
```

```
# %%
%matplotlib inline
import tensorflow as tf
import tensorflow.examples.tutorials.mnist.input_data as input_d
ata
import numpy as np
import matplotlib.pyplot as plt
```

```
# %%
# get the classic mnist dataset
# one-hot means a sparse vector for every observation where only
# the class label is 1, and every other class is 0.
# more info here:
# https://www.tensorflow.org/versions/0.6.0/tutorials/mnist/down
load/index.html#dataset-object
mnist = input_data.read_data_sets('MNIST_data/', one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

```
# %%
# mnist is now a DataSet with accessors for:
# 'train', 'test', and 'validation'.
# within each, we can access:
# images, labels, and num_examples
print(mnist.train.num_examples,
      mnist.test.num_examples,
      mnist.validation.num_examples)
```

```
(55000, 10000, 5000)
```

```
# %% the images are stored as:
# n_observations x n_features tensor (n-dim array)
# the labels are stored as n_observations x n_labels,
# where each observation is a one-hot vector.
print(mnist.train.images.shape, mnist.train.labels.shape)
```

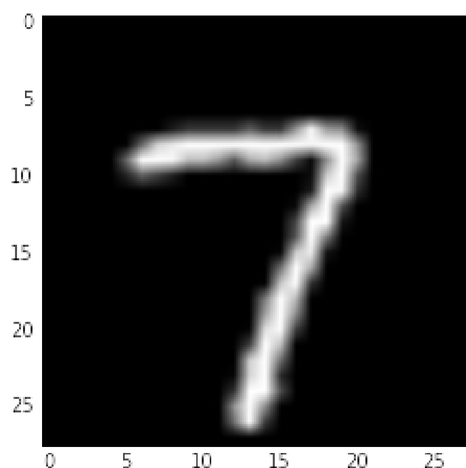
```
((55000, 784), (55000, 10))
```

```
# %% the range of the values of the images is from 0-1
print(np.min(mnist.train.images), np.max(mnist.train.images))
```

```
(0.0, 1.0)
```

```
# %% we can visualize any one of the images by reshaping it to a
 28x28 image
plt.imshow(np.reshape(mnist.train.images[100, :], (28, 28)), cma
p='gray')
```

```
<matplotlib.image.AxesImage at 0x7fc304bdf750>
```



```
# %% We can create a container for an input image using tensorfl
ow's graph:
# We allow the first dimension to be None, since this will event
ually
# represent our mini-batches, or how many images we feed into a
network
# at a time during training/validation/testing.
# The second dimension is the number of features that the image
has.
n_input = 784
n_output = 10
net_input = tf.placeholder(tf.float32, [None, n_input])
```

```
# %% We can write a simple regression (y = W*x + b) as:
W = tf.Variable(tf.zeros([n_input, n_output]))
b = tf.Variable(tf.zeros([n_output]))
net_output = tf.nn.softmax(tf.matmul(net_input, W) + b)
```

```
# %% We'll create a placeholder for the true output of the netwo
rk
y_true = tf.placeholder(tf.float32, [None, 10])
```

```
# %% And then write our loss function:
cross_entropy = -tf.reduce_sum(y_true * tf.log(net_output))
```

```
# %% This would equate each label in our one-hot vector between
the
# prediction and actual using the argmax as the predicted label
correct_prediction = tf.equal(
    tf.argmax(net_output, 1), tf.argmax(y_true, 1))
```

```
# %% And now we can look at the mean of our network's correct gu
esses
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

```
# %% We can tell the tensorflow graph to train w/ gradient desce
nt using
# our loss function and an input learning rate
optimizer = tf.train.GradientDescentOptimizer(
    0.01).minimize(cross_entropy)
```

```
# %% We now create a new session to actually perform the initial
ization the
# variables:
sess = tf.Session()
sess.run(tf.initialize_all_variables())
```

```
# %% Now actually do some training:
batch_size = 100
n_epochs = 10
for epoch_i in range(n_epochs):
    for batch_i in range(mnist.train.num_examples // batch_size)
:
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        sess.run(optimizer, feed_dict={
            net_input: batch_xs,
            y_true: batch_ys
        })
    print(sess.run(accuracy,
                   feed_dict={
                       net_input: mnist.validation.images,
                       y_true: mnist.validation.labels
                   }))
```

```
0.8994
0.9236
0.9242
0.9152
0.9216
0.915
0.918
0.9256
0.926
0.9228
```

```
# %% Print final test accuracy:
print(sess.run(accuracy,
               feed_dict={
                   net_input: mnist.test.images,
                   y_true: mnist.test.labels
               }))
```

```
0.923
```

```
# %%
"""
# We could do the same thing w/ Keras like so:
from keras.models import Sequential
model = Sequential()

from keras.layers.core import Dense, Activation
model.add(Dense(output_dim=10, input_dim=784, init='zero'))
model.add(Activation("softmax"))

from keras.optimizers import SGD
model.compile(loss='categorical_crossentropy',
    optimizer=SGD(lr=learning_rate))

model.fit(mnist.train.images, mnist.train.labels, nb_epoch=n_epo
chs,
          batch_size=batch_size, show_accuracy=True)

objective_score = model.evaluate(mnist.test.images, mnist.test.l
abels,
                                 batch_size=100, show_accuracy=T
rue)
"""
```

```
'\n# We could do the same thing w/ Keras like so:\nfrom keras.mo
dels import Sequential\nmodel = Sequential()\n\nfrom keras.layer
s.core import Dense, Activation\nmodel.add(Dense(output_dim=10,
input_dim=784, init=\'zero\'))\nmodel.add(Activation("softmax"))
\n\nfrom keras.optimizers import SGD\nmodel.compile(loss=\'categ
orical_crossentropy\', \n    optimizer=SGD(lr=learning_rate))\n\
nmodel.fit(mnist.train.images, mnist.train.labels, nb_epoch=n_ep
ochs,\n          batch_size=batch_size, show_accuracy=True)\n\no
bjective_score = model.evaluate(mnist.test.images, mnist.test.la
bels,\n                                 batch_size=100, show_acc
uracy=True)\n'
```

```
"""Simple tutorial following the TensorFlow example of a Convolu
tional Network.

Parag K. Mital, Jan. 2016"""
```

```
'Simple tutorial following the TensorFlow example of a Convoluti
onal Network.\n\nParag K. Mital, Jan. 2016'
```

```
# %% Imports
%matplotlib inline
import tensorflow as tf
import tensorflow.examples.tutorials.mnist.input_data as input_d
ata
from libs.utils import *
import matplotlib.pyplot as plt
```

```
# %% Setup input to the network and true output label.  These are

# simply placeholders which we'll fill in later.
mnist = input_data.read_data_sets('MNIST_data/', one_hot=True)
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

```
# %% Since x is currently [batch, height*width], we need to resh
ape to a
# 4-D tensor to use it in a convolutional graph.  If one compone
nt of
# `shape` is the special value -1, the size of that dimension is
# computed so that the total size remains constant.  Since we ha
ven't
# defined the batch dimension's shape yet, we use -1 to denote t
his
# dimension should not change size.
x_tensor = tf.reshape(x, [-1, 28, 28, 1])
```

```
# %% We'll setup the first convolutional layer
# Weight matrix is [height x width x input_channels x output_cha
nnels]
filter_size = 5
n_filters_1 = 16
W_conv1 = weight_variable([filter_size, filter_size, 1, n_filter
s_1])
```

```
# %% Bias is [output_channels]
b_conv1 = bias_variable([n_filters_1])
```

```
# %% Now we can build a graph which does the first layer of conv
olution:
# we define our stride as batch x height x width x channels
# instead of pooling, we use strides of 2 and more layers
# with smaller filters.
h_conv1 = tf.nn.relu(
    tf.nn.conv2d(input=x_tensor,
                 filter=W_conv1,
                 strides=[1, 2, 2, 1],
                 padding='SAME') +
    b_conv1)
```

```
# %% And just like the first layer, add additional layers to cre
ate
# a deep net
n_filters_2 = 16
W_conv2 = weight_variable([filter_size, filter_size, n_filters_1
, n_filters_2])
b_conv2 = bias_variable([n_filters_2])
h_conv2 = tf.nn.relu(
    tf.nn.conv2d(input=h_conv1,
                 filter=W_conv2,
                 strides=[1, 2, 2, 1],
                 padding='SAME') +
    b_conv2)
```

```
# %% We'll now reshape so we can connect to a fully-connected la
yer:
h_conv2_flat = tf.reshape(h_conv2, [-1, 7 * 7 * n_filters_2])
```

```
# %% Create a fully-connected layer:
n_fc = 1024
W_fc1 = weight_variable([7 * 7 * n_filters_2, n_fc])
b_fc1 = bias_variable([n_fc])
h_fc1 = tf.nn.relu(tf.matmul(h_conv2_flat, W_fc1) + b_fc1)
```

```
# %% We can add dropout for regularizing and to reduce overfitti
ng like so:
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

```
# %% And finally our softmax layer:
W_fc2 = weight_variable([n_fc, 10])
b_fc2 = bias_variable([10])
y_pred = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

```
# %% Define loss/eval/training functions
cross_entropy = -tf.reduce_sum(y * tf.log(y_pred))
optimizer = tf.train.AdamOptimizer().minimize(cross_entropy)
```

```
# %% Monitor accuracy
correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y,
1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
```

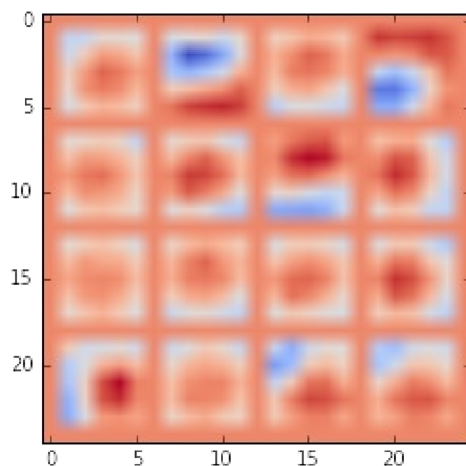
```
# %% We now create a new session to actually perform the initial
ization the
# variables:
sess = tf.Session()
sess.run(tf.initialize_all_variables())
```

```
# %% We'll train in minibatches and report accuracy:
batch_size = 100
n_epochs = 5
for epoch_i in range(n_epochs):
    for batch_i in range(mnist.train.num_examples // batch_size)
:
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        sess.run(optimizer, feed_dict={
            x: batch_xs, y: batch_ys, keep_prob: 0.5})
    print(sess.run(accuracy,
                   feed_dict={
                       x: mnist.validation.images,
                       y: mnist.validation.labels,
                       keep_prob: 1.0
                   }))
```

```
0.9624
0.9802
0.9828
0.9866
0.987
```

```
# %% Let's take a look at the kernels we've learned
W = sess.run(W_conv1)
plt.imshow(montage(W / np.max(W)), cmap='coolwarm')
```

```
<matplotlib.image.AxesImage at 0x7f04141f3390>
```



```
"""Tutorial on how to build a convnet w/ modern changes, e.g.
Batch Normalization, Leaky rectifiers, and strided convolution.

Parag K. Mital, Jan 2016.
"""
```

```
# %%
import tensorflow as tf
from libs.batch_norm import batch_norm
from libs.activations import lrelu
from libs.connections import conv2d, linear
from libs.datasets import MNIST
```

```
# %% Setup input to the network and true output label.  These ar
e
# simply placeholders which we'll fill in later.
mnist = MNIST()
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])
```

```
# %% We add a new type of placeholder to denote when we are trai
ning.
# This will be used to change the way we compute the network dur
ing
# training/testing.
is_training = tf.placeholder(tf.bool, name='is_training')
```

```
# %% We'll convert our MNIST vector data to a 4-D tensor:
# N x W x H x C
x_tensor = tf.reshape(x, [-1, 28, 28, 1])
```

```
# %% We'll use a new method called  batch normalization.
# This process attempts to "reduce internal covariate shift"
# which is a fancy way of saying that it will normalize updates
for each
# batch using a smoothed version of the batch mean and variance
# The original paper proposes using this before any nonlineariti
es
h_1 = lrelu(batch_norm(conv2d(x_tensor, 32, name='conv1'),
                       is_training, scope='bn1'), name='lrelu1')
h_2 = lrelu(batch_norm(conv2d(h_1, 64, name='conv2'),
                       is_training, scope='bn2'), name='lrelu2')
h_3 = lrelu(batch_norm(conv2d(h_2, 64, name='conv3'),
                       is_training, scope='bn3'), name='lrelu3')
h_3_flat = tf.reshape(h_3, [-1, 64 * 4 * 4])
h_4 = linear(h_3_flat, 10)
y_pred = tf.nn.softmax(h_4)
```

```
# %% Define loss/eval/training functions
cross_entropy = -tf.reduce_sum(y * tf.log(y_pred))
train_step = tf.train.AdamOptimizer().minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y,
 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
```

```
# %% We now create a new session to actually perform the initial
ization the
# variables:
sess = tf.Session()
sess.run(tf.initialize_all_variables())
```

```
# %% We'll train in minibatches and report accuracy:
n_epochs = 10
batch_size = 100
for epoch_i in range(n_epochs):
    for batch_i in range(mnist.train.num_examples // batch_size)
:
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        sess.run(train_step, feed_dict={
            x: batch_xs, y: batch_ys, is_training: True})
    print(sess.run(accuracy,
                   feed_dict={
                       x: mnist.validation.images,
                       y: mnist.validation.labels,
                       is_training: False
                   }))
```

```
"""Tutorial on how to create an autoencoder w/ Tensorflow.

Parag K. Mital, Jan 2016
"""
```

```
'Tutorial on how to create an autoencoder w/ Tensorflow.\n\nPara
g K. Mital, Jan 2016\n'
```

```
# %% Imports
%matplotlib inline
import tensorflow as tf
import numpy as np
import math
```

```
# %% Autoencoder definition
def autoencoder(dimensions=[784, 512, 256, 64]):
    """Build a deep autoencoder w/ tied weights.

    Parameters
    ----------
    dimensions : list, optional
        The number of neurons for each layer of the autoencoder.

    Returns
    -------
    x : Tensor
        Input placeholder to the network
    z : Tensor
        Inner-most latent representation
    y : Tensor
        Output reconstruction of the input
    cost : Tensor
        Overall cost to use for training
    """
    # %% input to the network
    x = tf.placeholder(tf.float32, [None, dimensions[0]], name='
x')
    current_input = x

    # %% Build the encoder
    encoder = []
    for layer_i, n_output in enumerate(dimensions[1:]):
        n_input = int(current_input.get_shape()[1])
        W = tf.Variable(
            tf.random_uniform([n_input, n_output],
                              -1.0 / math.sqrt(n_input),
                              1.0 / math.sqrt(n_input)))
```

```
        b = tf.Variable(tf.zeros([n_output]))
        encoder.append(W)
        output = tf.nn.tanh(tf.matmul(current_input, W) + b)
        current_input = output

    # %% latent representation
    z = current_input
    encoder.reverse()

    # %% Build the decoder using the same weights
    for layer_i, n_output in enumerate(dimensions[:-1][::-1]):
        W = tf.transpose(encoder[layer_i])
        b = tf.Variable(tf.zeros([n_output]))
        output = tf.nn.tanh(tf.matmul(current_input, W) + b)
        current_input = output

    # %% now have the reconstruction through the network
    y = current_input

    # %% cost function measures pixel-wise difference
    cost = tf.reduce_sum(tf.square(y - x))
    return {'x': x, 'z': z, 'y': y, 'cost': cost}
```

```
# %% Basic test
def test_mnist():
    """Test the autoencoder using MNIST."""
    import tensorflow as tf
    import tensorflow.examples.tutorials.mnist.input_data as inp
ut_data
    # from matplotlib import use
    # use('Agg')
    import matplotlib.pyplot as plt

    # %%
    # load MNIST as before
    mnist = input_data.read_data_sets('MNIST_data', one_hot=True
)
    mean_img = np.mean(mnist.train.images, axis=0)
    ae = autoencoder(dimensions=[784, 256, 64])

    # %%
    learning_rate = 0.001
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(a
e['cost'])

    # %%
    # We create a session to use the graph
    sess = tf.Session()
    sess.run(tf.initialize_all_variables())

    # %%
```

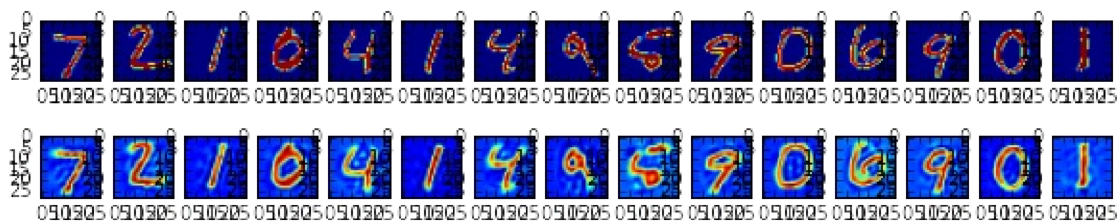
```
    # Fit all training data
    batch_size = 50
    n_epochs = 10
    for epoch_i in range(n_epochs):
        for batch_i in range(mnist.train.num_examples // batch_s
ize):
            batch_xs, _ = mnist.train.next_batch(batch_size)
            train = np.array([img - mean_img for img in batch_xs
])
            sess.run(optimizer, feed_dict={ae['x']: train})
        print(epoch_i, sess.run(ae['cost'], feed_dict={ae['x']:
train}))

    # %%
    # Plot example reconstructions
    n_examples = 15
    test_xs, _ = mnist.test.next_batch(n_examples)
    test_xs_norm = np.array([img - mean_img for img in test_xs])
    recon = sess.run(ae['y'], feed_dict={ae['x']: test_xs_norm})
    fig, axs = plt.subplots(2, n_examples, figsize=(10, 2))
    for example_i in range(n_examples):
        axs[0][example_i].imshow(
            np.reshape(test_xs[example_i, :], (28, 28)))
        axs[1][example_i].imshow(
            np.reshape([recon[example_i, :] + mean_img], (28, 28
)))
    fig.show()
    plt.draw()
```

```
# %%
if __name__ == '__main__':
    test_mnist()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
(0, 554.17175)
(1, 451.15411)
(2, 420.77158)
(3, 437.03281)
(4, 409.98288)
(5, 363.77893)
(6, 420.08453)
(7, 396.18784)
(8, 365.31839)
(9, 407.51379)


/home/heythisischo/anaconda2/lib/python2.7/site-packages/matplot
lib/figure.py:397: UserWarning: matplotlib is currently using a
non-GUI backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



```
"""Tutorial on how to create a denoising autoencoder w/ Tensorfl
ow.

Parag K. Mital, Jan 2016
"""
%matplotlib inline
import tensorflow as tf
import numpy as np
import math
from libs.utils import corrupt
```

```
# %%
def autoencoder(dimensions=[784, 512, 256, 64]):
    """Build a deep denoising autoencoder w/ tied weights.

    Parameters
    ----------
    dimensions : list, optional
        The number of neurons for each layer of the autoencoder.

    Returns
    -------
    x : Tensor
        Input placeholder to the network
    z : Tensor
        Inner-most latent representation
    y : Tensor
        Output reconstruction of the input
    cost : Tensor
        Overall cost to use for training
    """
    # input to the network
    x = tf.placeholder(tf.float32, [None, dimensions[0]], name='
x')

    # Probability that we will corrupt input.
    # This is the essence of the denoising autoencoder, and is p
retty
    # basic.  We'll feed forward a noisy input, allowing our net
work
    # to generalize better, possibly, to occlusions of what we're

    # really interested in.  But to measure accuracy, we'll still

    # enforce a training signal which measures the original imag
e's
    # reconstruction cost.
    #
    # We'll change this to 1 during training
    # but when we're ready for testing/production ready environm
```

```
ents,
    # we'll put it back to 0.
    corrupt_prob = tf.placeholder(tf.float32, [1])
    current_input = corrupt(x) * corrupt_prob + x * (1 - corrupt
_prob)

    # Build the encoder
    encoder = []
    for layer_i, n_output in enumerate(dimensions[1:]):
        n_input = int(current_input.get_shape()[1])
        W = tf.Variable(
            tf.random_uniform([n_input, n_output],
                              -1.0 / math.sqrt(n_input),
                              1.0 / math.sqrt(n_input)))
        b = tf.Variable(tf.zeros([n_output]))
        encoder.append(W)
        output = tf.nn.tanh(tf.matmul(current_input, W) + b)
        current_input = output
    # latent representation
    z = current_input
    encoder.reverse()
    # Build the decoder using the same weights
    for layer_i, n_output in enumerate(dimensions[:-1][::-1]):
        W = tf.transpose(encoder[layer_i])
        b = tf.Variable(tf.zeros([n_output]))
        output = tf.nn.tanh(tf.matmul(current_input, W) + b)
        current_input = output
    # now have the reconstruction through the network
    y = current_input
    # cost function measures pixel-wise difference
    cost = tf.sqrt(tf.reduce_mean(tf.square(y - x)))
    return {'x': x, 'z': z, 'y': y,
            'corrupt_prob': corrupt_prob,
            'cost': cost}
```

```
# %% Basic test
def test_mnist():
    import tensorflow as tf
    import tensorflow.examples.tutorials.mnist.input_data as inp
ut_data
    import matplotlib.pyplot as plt

    # %%
    # load MNIST as before
    mnist = input_data.read_data_sets('MNIST_data', one_hot=True
)
    mean_img = np.mean(mnist.train.images, axis=0)
    ae = autoencoder(dimensions=[784, 256, 64])

    # %%
```

```
    learning_rate = 0.001
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(a
e['cost'])

    # %%
    # We create a session to use the graph
    sess = tf.Session()
    sess.run(tf.initialize_all_variables())

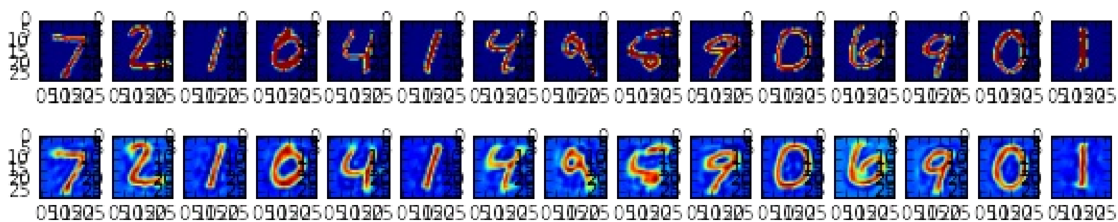
    # %%
    # Fit all training data
    batch_size = 50
    n_epochs = 10
    for epoch_i in range(n_epochs):
        for batch_i in range(mnist.train.num_examples // batch_s
ize):
            batch_xs, _ = mnist.train.next_batch(batch_size)
            train = np.array([img - mean_img for img in batch_xs
])
            sess.run(optimizer, feed_dict={
                ae['x']: train, ae['corrupt_prob']: [1.0]})
        print(epoch_i, sess.run(ae['cost'], feed_dict={
            ae['x']: train, ae['corrupt_prob']: [1.0]}))

    # %%
    # Plot example reconstructions
    n_examples = 15
    test_xs, _ = mnist.test.next_batch(n_examples)
    test_xs_norm = np.array([img - mean_img for img in test_xs])
    recon = sess.run(ae['y'], feed_dict={
        ae['x']: test_xs_norm, ae['corrupt_prob']: [0.0]})
    fig, axs = plt.subplots(2, n_examples, figsize=(10, 2))
    for example_i in range(n_examples):
        axs[0][example_i].imshow(
            np.reshape(test_xs[example_i, :], (28, 28)))
        axs[1][example_i].imshow(
            np.reshape([recon[example_i, :] + mean_img], (28, 28
)))
    fig.show()
    plt.draw()
```

```
if __name__ == '__main__':
    test_mnist()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
(0, 0.14199062)
(1, 0.13837112)
(2, 0.1353873)
(3, 0.13011663)
(4, 0.13082016)
(5, 0.12943956)
(6, 0.13008749)
(7, 0.12971884)
(8, 0.12277839)
(9, 0.12314773)


/home/heythisischo/anaconda2/lib/python2.7/site-packages/matplot
lib/figure.py:397: UserWarning: matplotlib is currently using a
non-GUI backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



```
"""Tutorial on how to create a convolutional autoencoder w/ Tens
orflow.

Parag K. Mital, Jan 2016
"""
# %% imports
%matplotlib inline
import tensorflow as tf
import numpy as np
import math
from libs.activations import lrelu
from libs.utils import corrupt
```

```
# %%
def autoencoder(input_shape=[None, 784],
                n_filters=[1, 10, 10, 10],
                filter_sizes=[3, 3, 3, 3],
                corruption=False):
    """Build a deep denoising autoencoder w/ tied weights.

    Parameters
    ----------
    input_shape : list, optional
        Description
    n_filters : list, optional
        Description
    filter_sizes : list, optional
        Description

    Returns
    -------
    x : Tensor
        Input placeholder to the network
    z : Tensor
        Inner-most latent representation
    y : Tensor
        Output reconstruction of the input
    cost : Tensor
        Overall cost to use for training

    Raises
    ------
    ValueError
        Description
    """
    # %%
    # input to the network
    x = tf.placeholder(
        tf.float32, input_shape, name='x')
```

```
    # %%
    # Optionally apply denoising autoencoder
    if corruption:
        x_noise = corrupt(x)
    else:
        x_noise = x

    # %%
    # ensure 2-d is converted to square tensor.
    if len(x.get_shape()) == 2:
        x_dim = np.sqrt(x_noise.get_shape().as_list()[1])
        if x_dim != int(x_dim):
            raise ValueError('Unsupported input dimensions')
        x_dim = int(x_dim)
        x_tensor = tf.reshape(
            x_noise, [-1, x_dim, x_dim, n_filters[0]])
    elif len(x_noise.get_shape()) == 4:
        x_tensor = x_noise
    else:
        raise ValueError('Unsupported input dimensions')
    current_input = x_tensor

    # %%
    # Build the encoder
    encoder = []
    shapes = []
    for layer_i, n_output in enumerate(n_filters[1:]):
        n_input = current_input.get_shape().as_list()[3]
        shapes.append(current_input.get_shape().as_list())
        W = tf.Variable(
            tf.random_uniform([
                filter_sizes[layer_i],
                filter_sizes[layer_i],
                n_input, n_output],
                -1.0 / math.sqrt(n_input),
                1.0 / math.sqrt(n_input)))
        b = tf.Variable(tf.zeros([n_output]))
        encoder.append(W)
        output = lrelu(
            tf.add(tf.nn.conv2d(
                current_input, W, strides=[1, 2, 2, 1], padding=
'SAME'), b))
        current_input = output

    # %%
    # store the latent representation
    z = current_input
    encoder.reverse()
    shapes.reverse()

    # %%
    # Build the decoder using the same weights
    for layer_i, shape in enumerate(shapes):
```

```
        W = encoder[layer_i]
        b = tf.Variable(tf.zeros([W.get_shape().as_list()[2]]))
        output = lrelu(tf.add(
            tf.nn.conv2d_transpose(
                current_input, W,
                tf.pack([tf.shape(x)[0], shape[1], shape[2], sha
pe[3]]),
                strides=[1, 2, 2, 1], padding='SAME'), b))
        current_input = output

    # %%
    # now have the reconstruction through the network
    y = current_input
    # cost function measures pixel-wise difference
    cost = tf.reduce_sum(tf.square(y - x_tensor))

    # %%
    return {'x': x, 'z': z, 'y': y, 'cost': cost}
```

```
# %%
def test_mnist():
    """Test the convolutional autoencder using MNIST."""
    # %%
    import tensorflow as tf
    import tensorflow.examples.tutorials.mnist.input_data as inp
ut_data
    import matplotlib.pyplot as plt

    # %%
    # load MNIST as before
    mnist = input_data.read_data_sets('MNIST_data', one_hot=True
)
    mean_img = np.mean(mnist.train.images, axis=0)
    ae = autoencoder()

    # %%
    learning_rate = 0.01
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(a
e['cost'])

    # %%
    # We create a session to use the graph
    sess = tf.Session()
    sess.run(tf.initialize_all_variables())

    # %%
    # Fit all training data
    batch_size = 100
    n_epochs = 10
    for epoch_i in range(n_epochs):
        for batch_i in range(mnist.train.num_examples // batch_s
```

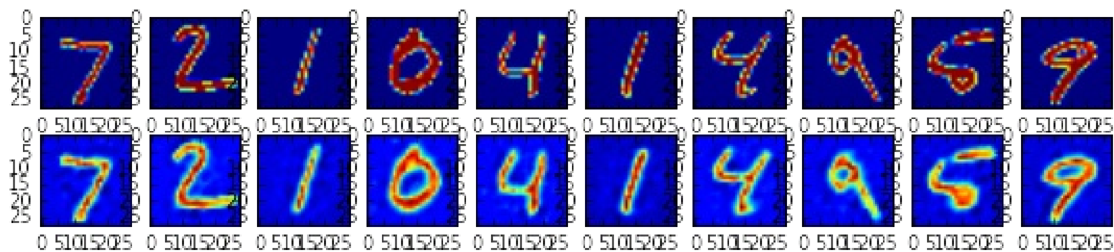
```
ize):
            batch_xs, _ = mnist.train.next_batch(batch_size)
            train = np.array([img - mean_img for img in batch_xs
])
            sess.run(optimizer, feed_dict={ae['x']: train})
        print(epoch_i, sess.run(ae['cost'], feed_dict={ae['x']:
train}))

    # %%
    # Plot example reconstructions
    n_examples = 10
    test_xs, _ = mnist.test.next_batch(n_examples)
    test_xs_norm = np.array([img - mean_img for img in test_xs])
    recon = sess.run(ae['y'], feed_dict={ae['x']: test_xs_norm})
    print(recon.shape)
    fig, axs = plt.subplots(2, n_examples, figsize=(10, 2))
    for example_i in range(n_examples):
        axs[0][example_i].imshow(
            np.reshape(test_xs[example_i, :], (28, 28)))
        axs[1][example_i].imshow(
            np.reshape(
                np.reshape(recon[example_i, ...], (784,)) + mean
_img,
                (28, 28)))
    fig.show()
    plt.draw()
```

```
# %%
if __name__ == '__main__':
    test_mnist()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
(0, 1033.2483)
(1, 780.23218)
(2, 721.82605)
(3, 761.09747)
(4, 667.92712)
(5, 709.97705)
(6, 652.50287)
(7, 691.22681)
(8, 686.44324)
(9, 662.02979)
(10, 28, 28, 1)


/home/heythisischo/anaconda2/lib/python2.7/site-packages/matplot
lib/figure.py:397: UserWarning: matplotlib is currently using a
non-GUI backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



```
"""In progress.

Parag K. Mital, Jan 2016.
"""
```

```
# %%
import tensorflow as tf
from libs.connections import conv2d, linear
from collections import namedtuple
from math import sqrt
```

```
# %%
def residual_network(x, n_outputs,
                     activation=tf.nn.relu):
    """Builds a residual network.

    Parameters
    ----------
    x : Placeholder
        Input to the network
    n_outputs : TYPE
        Number of outputs of final softmax
    activation : Attribute, optional
        Nonlinearity to apply after each convolution

    Returns
    -------
    net : Tensor
        Description

    Raises
    ------
    ValueError
        If a 2D Tensor is input, the Tensor must be square or el
se
        the network can't be converted to a 4D Tensor.
    """
    # %%
    LayerBlock = namedtuple(
        'LayerBlock', ['num_repeats', 'num_filters', 'bottleneck
_size'])
    blocks = [LayerBlock(3, 128, 32),
              LayerBlock(3, 256, 64),
              LayerBlock(3, 512, 128),
              LayerBlock(3, 1024, 256)]

    # %%
    input_shape = x.get_shape().as_list()
    if len(input_shape) == 2:
```

```
        ndim = int(sqrt(input_shape[1]))
        if ndim * ndim != input_shape[1]:
            raise ValueError('input_shape should be square')
        x = tf.reshape(x, [-1, ndim, ndim, 1])

    # %%
    # First convolution expands to 64 channels and downsamples
    net = conv2d(x, 64, k_h=7, k_w=7,
                 name='conv1',
                 activation=activation)

    # %%
    # Max pool and downsampling
    net = tf.nn.max_pool(
        net, [1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')

    # %%
    # Setup first chain of resnets
    net = conv2d(net, blocks[0].num_filters, k_h=1, k_w=1,
                 stride_h=1, stride_w=1, padding='VALID', name='
conv2')

    # %%
    # Loop through all res blocks
    for block_i, block in enumerate(blocks):
        for repeat_i in range(block.num_repeats):

            name = 'block_%d/repeat_%d' % (block_i, repeat_i)
            conv = conv2d(net, block.bottleneck_size, k_h=1, k_w
=1,
                          padding='VALID', stride_h=1, stride_w=
1,
                          activation=activation,
                          name=name + '/conv_in')

            conv = conv2d(conv, block.bottleneck_size, k_h=3, k_
w=3,
                          padding='SAME', stride_h=1, stride_w=1
,
                          activation=activation,
                          name=name + '/conv_bottleneck')

            conv = conv2d(conv, block.num_filters, k_h=1, k_w=1,
                          padding='VALID', stride_h=1, stride_w=
1,
                          activation=activation,
                          name=name + '/conv_out')

            net = conv + net
        try:
            # upscale to the next block size
            next_block = blocks[block_i + 1]
            net = conv2d(net, next_block.num_filters, k_h=1, k_w
```

```
=1,
                         padding='SAME', stride_h=1, stride_w=1,
 bias=False,
                         name='block_%d/conv_upscale' % block_i)
        except IndexError:
            pass

    # %%
    net = tf.nn.avg_pool(net,
                         ksize=[1, net.get_shape().as_list()[1],
                                net.get_shape().as_list()[2], 1]
,
                         strides=[1, 1, 1, 1], padding='VALID')
    net = tf.reshape(
        net,
        [-1, net.get_shape().as_list()[1] *
         net.get_shape().as_list()[2] *
         net.get_shape().as_list()[3]])

    net = linear(net, n_outputs, activation=tf.nn.softmax)

    # %%
    return net


def test_mnist():
    """Test the resnet on MNIST."""
    import tensorflow.examples.tutorials.mnist.input_data as inp
ut_data

    mnist = input_data.read_data_sets('MNIST_data/', one_hot=Tru
e)
    x = tf.placeholder(tf.float32, [None, 784])
    y = tf.placeholder(tf.float32, [None, 10])
    y_pred = residual_network(x, 10)

    # %% Define loss/eval/training functions
    cross_entropy = -tf.reduce_sum(y * tf.log(y_pred))
    optimizer = tf.train.AdamOptimizer().minimize(cross_entropy)

    # %% Monitor accuracy
    correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argma
x(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float
'))

    # %% We now create a new session to actually perform the ini
tialization the
    # variables:
    sess = tf.Session()
    sess.run(tf.initialize_all_variables())

    # %% We'll train in minibatches and report accuracy:
```

```
    batch_size = 50
    n_epochs = 5
    for epoch_i in range(n_epochs):
        # Training
        train_accuracy = 0
        for batch_i in range(mnist.train.num_examples // batch_s
ize):
            batch_xs, batch_ys = mnist.train.next_batch(batch_si
ze)
            train_accuracy += sess.run([optimizer, accuracy], fe
ed_dict={
                x: batch_xs, y: batch_ys})[1]
        train_accuracy /= (mnist.train.num_examples // batch_siz
e)

        # Validation
        valid_accuracy = 0
        for batch_i in range(mnist.validation.num_examples // ba
tch_size):
            batch_xs, batch_ys = mnist.validation.next_batch(bat
ch_size)
            valid_accuracy += sess.run(accuracy,
                                       feed_dict={
                                           x: batch_xs,
                                           y: batch_ys
                                       })
        valid_accuracy /= (mnist.validation.num_examples // batc
h_size)
        print('epoch:', epoch_i, ', train:',
              train_accuracy, ', valid:', valid_accuracy)


if __name__ == '__main__':
    test_mnist()
```

```
"""Training a variational autoencoder with 2 layer fully-connect
ed
encoder/decoders and gaussian noise distribution.

Parag K. Mital, Jan 2016
"""
import tensorflow as tf
import numpy as np
from libs.utils import weight_variable, bias_variable, montage_b
atch
```

```
# %%
def VAE(input_shape=[None, 784],
        n_components_encoder=2048,
        n_components_decoder=2048,
        n_hidden=2,
        debug=False):
    # %%
    # Input placeholder
    if debug:
        input_shape = [50, 784]
        x = tf.Variable(np.zeros((input_shape), dtype=np.float32
))
    else:
        x = tf.placeholder(tf.float32, input_shape)

    activation = tf.nn.softplus

    dims = x.get_shape().as_list()
    n_features = dims[1]

    W_enc1 = weight_variable([n_features, n_components_encoder])
    b_enc1 = bias_variable([n_components_encoder])
    h_enc1 = activation(tf.matmul(x, W_enc1) + b_enc1)

    W_enc2 = weight_variable([n_components_encoder, n_components
_encoder])
    b_enc2 = bias_variable([n_components_encoder])
    h_enc2 = activation(tf.matmul(h_enc1, W_enc2) + b_enc2)

    W_enc3 = weight_variable([n_components_encoder, n_components
_encoder])
    b_enc3 = bias_variable([n_components_encoder])
    h_enc3 = activation(tf.matmul(h_enc2, W_enc3) + b_enc3)

    W_mu = weight_variable([n_components_encoder, n_hidden])
    b_mu = bias_variable([n_hidden])

    W_log_sigma = weight_variable([n_components_encoder, n_hidde
n])
```

```
    b_log_sigma = bias_variable([n_hidden])

    z_mu = tf.matmul(h_enc3, W_mu) + b_mu
    z_log_sigma = 0.5 * (tf.matmul(h_enc3, W_log_sigma) + b_log_
sigma)

    # %%
    # Sample from noise distribution p(eps) ~ N(0, 1)
    if debug:
        epsilon = tf.random_normal(
            [dims[0], n_hidden])
    else:
        epsilon = tf.random_normal(
            tf.pack([tf.shape(x)[0], n_hidden]))

    # Sample from posterior
    z = z_mu + tf.exp(z_log_sigma) * epsilon

    W_dec1 = weight_variable([n_hidden, n_components_decoder])
    b_dec1 = bias_variable([n_components_decoder])
    h_dec1 = activation(tf.matmul(z, W_dec1) + b_dec1)

    W_dec2 = weight_variable([n_components_decoder, n_components
_decoder])
    b_dec2 = bias_variable([n_components_decoder])
    h_dec2 = activation(tf.matmul(h_dec1, W_dec2) + b_dec2)

    W_dec3 = weight_variable([n_components_decoder, n_components
_decoder])
    b_dec3 = bias_variable([n_components_decoder])
    h_dec3 = activation(tf.matmul(h_dec2, W_dec3) + b_dec3)

    W_mu_dec = weight_variable([n_components_decoder, n_features
])
    b_mu_dec = bias_variable([n_features])
    y = tf.nn.tanh(tf.matmul(h_dec3, W_mu_dec) + b_mu_dec)

    # p(x|z)
    log_px_given_z = -tf.reduce_sum(
        x * tf.log(y + 1e-10) +
        (1 - x) * tf.log(1 - y + 1e-10), 1)

    # d_kl(q(z|x)||p(z))
    # Appendix B: 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    kl_div = -0.5 * tf.reduce_sum(
        1.0 + 2.0 * z_log_sigma - tf.square(z_mu) - tf.exp(2.0 *
 z_log_sigma),
        1)
    loss = tf.reduce_mean(log_px_given_z + kl_div)

    return {'cost': loss, 'x': x, 'z': z, 'y': y}
```

```
# %%
def test_mnist():
    """Summary

    Returns
    -------
    name : TYPE
        Description
    """
    # %%
    import tensorflow as tf
    import tensorflow.examples.tutorials.mnist.input_data as inp
ut_data
    import matplotlib.pyplot as plt

    # %%
    # load MNIST as before
    mnist = input_data.read_data_sets('MNIST_data', one_hot=True
)
    ae = VAE()

    # %%
    learning_rate = 0.001
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(a
e['cost'])

    # %%
    # We create a session to use the graph
    sess = tf.Session()
    sess.run(tf.initialize_all_variables())

    # %%
    # Fit all training data
    t_i = 0
    batch_size = 100
    n_epochs = 50
    n_examples = 20
    test_xs, _ = mnist.test.next_batch(n_examples)
    xs, ys = mnist.test.images, mnist.test.labels
    fig_manifold, ax_manifold = plt.subplots(1, 1)
    fig_reconstruction, axs_reconstruction = plt.subplots(2, n_e
xamples, figsize=(10, 2))
    fig_image_manifold, ax_image_manifold = plt.subplots(1, 1)
    for epoch_i in range(n_epochs):
        print('--- Epoch', epoch_i)
        train_cost = 0
        for batch_i in range(mnist.train.num_examples // batch_s
ize):
            batch_xs, _ = mnist.train.next_batch(batch_size)
            train_cost += sess.run([ae['cost'], optimizer],
                                   feed_dict={ae['x']: batch_xs}
)[0]
            if batch_i % 2 == 0:
```

```
                # %%
                # Plot example reconstructions from latent layer
                imgs = []
                for img_i in np.linspace(-3, 3, n_examples):
                    for img_j in np.linspace(-3, 3, n_examples):
                        z = np.array([[img_i, img_j]], dtype=np.
float32)
                        recon = sess.run(ae['y'], feed_dict={ae[
'z']: z})
                        imgs.append(np.reshape(recon, (1, 28, 28
, 1)))
                imgs_cat = np.concatenate(imgs)
                ax_manifold.imshow(montage_batch(imgs_cat))
                fig_manifold.savefig('manifold_%08d.png' % t_i)

                # %%
                # Plot example reconstructions
                recon = sess.run(ae['y'], feed_dict={ae['x']: te
st_xs})
                print(recon.shape)
                for example_i in range(n_examples):
                    axs_reconstruction[0][example_i].imshow(
                        np.reshape(test_xs[example_i, :], (28, 2
8)),
                        cmap='gray')
                    axs_reconstruction[1][example_i].imshow(
                        np.reshape(
                            np.reshape(recon[example_i, ...], (7
84,)),
                            (28, 28)),
                        cmap='gray')
                    axs_reconstruction[0][example_i].axis('off')
                    axs_reconstruction[1][example_i].axis('off')
                fig_reconstruction.savefig('reconstruction_%08d.
png' % t_i)

                # %%
                # Plot manifold of latent layer
                zs = sess.run(ae['z'], feed_dict={ae['x']: xs})
                ax_image_manifold.clear()
                ax_image_manifold.scatter(zs[:, 0], zs[:, 1],
                    c=np.argmax(ys, 1), alpha=0.2)
                ax_image_manifold.set_xlim([-6, 6])
                ax_image_manifold.set_ylim([-6, 6])
                ax_image_manifold.axis('off')
                fig_image_manifold.savefig('image_manifold_%08d.
png' % t_i)

                t_i += 1


        print('Train cost:', train_cost /
              (mnist.train.num_examples // batch_size))
```

```
        valid_cost = 0
        for batch_i in range(mnist.validation.num_examples // ba
tch_size):
            batch_xs, _ = mnist.validation.next_batch(batch_size
)
            valid_cost += sess.run([ae['cost']],
                                   feed_dict={ae['x']: batch_xs}
)[0]
        print('Validation cost:', valid_cost /
              (mnist.validation.num_examples // batch_size))


if __name__ == '__main__':
    test_mnist()
```